Rešenja prvog kolokvijuma iz Operativnih sistema 2 Septembar 2014.

1. (10 poena)

```
const int MAXPRI = …, PRIRANGE = …, PRISTEP = 1;
extern PCB* idle;

#define max(a,b) (((a)>=(b))?(a):(b))
#define min(a,b) (((a)<=(b))?(a):(b))

class Scheduler {
public:
  Scheduler ();
  PCB* get ();
  void put (PCB*, int wasBlocked);
private:
  PCB* head[MAXPRI+1];
  PCB* tail[MAXPRI+1];
  int maxPri;
};

Scheduler::Scheduler () : maxPri(-1) {
  for (int i=0; i<MAXPRI+1; i++)
    head[i]=tail[i]=0;
}

void Scheduler::put (PCB* pcb, int wasBlocked) {
  if (pcb==0) return; // Exception!
  // Set new dynamic priority:
  int p = pcb->priority;
  if (wasBlocked) {
    int mp = min(MAXPRI,pcb->defaultPri+PRIRANGE);
    p = min(p+PRISTEP,mp);
  } else {
    int mp = max(0,pcb->defaultPri-PRIRANGE);
    p = max(p-PRISTEP,mp);
  }
  pcb->priority = p;
  // Put pcb in the corresponding slot:
  pcb->next = 0;
  if (tail[p]==0)
    tail[p] = head[p] = pcb;
  else
    tail[p] = tail[p]->next = pcb;
  if (p>maxPri) maxPri=p;
}

PCB* Scheduler::get () {
  if (maxPri==-1) return idle;
  PCB* ret = head[maxPri];
  head[maxPri] = head[maxPri]->next;
  if (head[maxPri]==0) {
    tail[maxPri]=0;
    while (maxPri>=0 && head[maxPri]==0) maxPri--;
  }
  ret->next = 0;
  return ret;
}
```

2. (10 poena)

```
monitor Event;
  export wait, signal;

  var flag : boolean,
      cond : condition;

  procedure wait ();
  begin
    while not flag do cond.wait;
    flag:=false;
  end;

  procedure signal ();
  begin
    flag:=true;
    cond.signal;
  end;

begin
  flag:=false;
end; (* Event *)
```

3. (10 poena)

```
 static final int N = ...;
 static final int M =...;
 static int count = N;
 static ArrayList<LinkedList<Socket>> blockedList = new
ArrayList<LinkedList<Socket>>(M);

 public static void main(String[] args) {
  for (int i = 0; i < M; i++) {
   blockedList.add(i, new LinkedList<Socket>());
  }
  try {
   ServerSocket sock = new ServerSocket(1033);
   while (true) {
    Socket clientSocket = sock.accept();
    BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    String msg = in.readLine();

    StringTokenizer st = new StringTokenizer(msg, "#");
    String request = st.nextToken();

    if (request.equals("Enter")) {
     if(count>0){
      sendMsgToClient(clientSocket, "Continue");
      count--;
     }else {
      int strNo = Integer.parseInt(st.nextToken());
      blockedList.get(strNo).addLast(clientSocket);
     }
    }else if (request.equals("Exit")) {
     boolean found = false;
     for (int i = 0; i < M; i++) {
       if(!blockedList.get(i).isEmpty()){
        sendMsgToClient(blockedList.get(i).poll(), "Continue");
        found = true;
        break;
       }
```

```
     }
     if (!found) count++;
    }
   }
  } catch (Exception e) { System.err.println(e);}
 }

 static void sendMsgToClient(Socket clientSocket,String msg) throws
UnknownHostException, IOException {
  PrintWriter newOut = new
PrintWriter(clientSocket.getOutputStream(),true);
  newOut.println(msg);
  clientSocket.close();
 }
}

// Client
public class Client {
 public static void main(String[] args) {
  try {
   while (true) {
    Socket srvSocket = new Socket("localhost", 1033);
    sendMsq(srvSocket, "#Enter#"+args[0]+"#"); // args[0] is street no.
    BufferedReader in = new BufferedReader(new
InputStreamReader(srvSocket.getInputStream()));
    System.out.println(in.readLine());
    srvSocket.close();

    //drive...
    Thread.sleep(1000);

    srvSocket = new Socket("localhost", 1033);
    sendMsq(srvSocket,"Exit");
    srvSocket.close();
   }
  } catch (Exception e) {
   System.err.println(e);
  }
 }
 private static void sendMsq(Socket srvSocket, String msg) throws
UnknownHostException, IOException {
  PrintWriter out = new PrintWriter(srvSocket.getOutputStream(), true);
  out.println(msg);
  }
}
```